LOGIC PROGRAMMING WITH NARROWING

AUTHOR(S):

Yamamoto, Akihiro

CITATION:

Yamamoto, Akihiro. LOGIC PROGRAMMING WITH NARROWING. 数理解析研究所講究録 1987, 618: 16-39

ISSUE DATE:

1987-04

URL:

http://hdl.handle.net/2433/99864

数理解析研究所講究録
618巻 1987年 16-39

16

LOGIC PROGRAMMING WITH NARROWING

Akihiro Yamamoto (山本 章博)

Department of Information Systems
Interdisciplinary Graduate School of Engineering Sciences
Kyushu University
6-1, Kasuga-koen, Kasuga, Fukuoka, 816 JAPAN

Abstract

In this paper we introduce a narrowing procedure into logic programming in order to treat equational theories. We give a refutation procedure for a program containing definite clauses, equations and axioms for equations. The refutation consists of SLD-resolution and narrowing so that they are treated equally. We give the fixpoint semantics of the refutation procedure and discuss the relation between the fixpoint and the least Herbrand model. We discuss the completeness of the refutation under some conditions.

1. Introduction

In this paper we introduce a narrowing procedure into logic programming in order to treat equational theories. The narrowing is the fundamental procedure which is used to enumulate a complete set of unifiers with respect to an equational theory.

There are many extended PROLOG systems making use of generalized unification (e.g., Kornfeld [10], Goguen and Meseguer [3]). Jaffer et al. [7] have given a general framework of the operational semantics and declarative semantics for refutation systems. Thus one may consider that we can extend standard PROLOG system by introducing some equational theories and replacing the algorithm for mgu by that for complete sets of unifiers. Even if we have implemented the extended system with the same searching strategy as that of standard PROLOG, the strategy might not halt because the complete set usually con-

tains infinitely many unifiers. The generalized unification with respect to an equational theory is an inference of the theory, and thus it should be treated on the same level of SLD-resolution. The intended system above cannot reflect this point. In terms of declarative semantics, this problem corresponds to the fact that we can identify a congruence class in the congruence space of the Herbrand base, but cannot identify a suitable element of the class. We need to clarify the process of constructing the congruence classes.

In the present paper, we discuss a system in which both SLD-resolution and narrowing are treated equally. In the system a logic program consists of equations and definite clauses. The definite clauses in a logic program should be transformed into their homogeneous forms, and axioms for equations are added to the program in order to describe the generalized unification. In van Emden and Lloyd [16], they have shown that the axiom for equations after the homogeneous transformation is the identity axiom for standard PROLOG. Hullot [6] has also shown that if the set of equations is canonical in the sense of term rewriting system, the complete set of unifiers can be enumulated by using narrowing and the identity axiom. Thus we need not extend the axioms for equations to introduce narrowing into SLD-resolution. Based on this consideration, we define a program which consists of homogeneous definite clauses, equations, and the identity axiom. We also give the fixpoint semantics of the program using Herbrand base, and discuss the completeness of the refutation procedure.

This paper is organized as follows. In the next section, we discuss the problem arising when we introduce generalized unification into logic programming. In Section 3 we explain the term rewriting system and narrowing. In Section 4, we define a refutation whose inference rules are narrowing and SLD-resolution, and show its soundness. In Section 5, we give a fixpoint semantics using Herbrand base. In Section 6, we discuss the completeness of refutation when the set of equations in program are canonical in the sense of a term rewriting system. In Section 7, we present an example of simple implementation.

2. Logic Programming and Generalized Unification

Let L be a first order language. We use the symbols, Π, Σ, V to denote the sets of predicate symbols, function symbols, variable symbols, respectively. The set Π contains the symbol '='. A(L), and B(L) denote the sets of atoms and ground atoms, respectively. $T(\Sigma \cup V)$ and $T(\Sigma)$ denote the sets of terms and ground terms, respectively. A word is an atom or a term. The set of variables occurring in a word t is denoted by V(t). We adopt the list-notation of DEC-10 PROLOG.

A substitution θ is a mapping from V into $T(\Sigma \cup V)$ such that the set $\{X \in V;\ X\theta \neq X\}$, denoted by $D(\theta)$, is finite. $D(\theta)$ is called the domain of θ. If $D(\theta)=\{X_1,\ldots,X_n\}$ and $X_i\theta = t_i$, then θ is denoted by a set $\{X_1 \leftarrow t_1,\ldots,X_n \leftarrow t_n\}$. The set $\cup_{i=1}^{n} V(t_i)$ is denoted by $I(\theta)$. $I(\theta)$ is the set of variables introduced by θ. Moreover, given a set of variables U, we define a substitution $\theta|_U$ by $\{X_i \leftarrow t_i;\ X_i \in U\}$.

A definite clause is a clause of the form $A \leftarrow B_1,\ldots,B_n$. A is called the head of the clause, and the sequence $B_1,\ldots,B_n$ is called its body. A logic program is a set of definite clauses in which the symbol '=' does not occur. A goal clause is a clause of the form $\leftarrow A_1,\ldots,A_k$. An equation is an atom which has the predicate symbol '='. For the sake of convenience, we also call a clause of the form $s=t \leftarrow$ an equation.

In this paper, we define the generalized unification with respect to an equational theory. We give the inference rules of the equational theory in a form of a set of definite clauses. We call them the equality axioms.

$$
\begin{aligned}
EQ = \ & \{E_I:\ X=X \leftarrow \\
& \cup \{E_S:\ Y=X \leftarrow X=Y\} \\
& \cup \{E_T:\ X=Z \leftarrow X=Y, Y=Z\} \\
& \cup \{E_f:\ f(X_1,\ldots,X_m)=f(Y_1,\ldots,Y_m) \leftarrow X_1=Y_1,\ldots,X_m=Y_m;\ f \in \Sigma\} \\
& \cup \{E_p:\ p(X_1,\ldots,X_m) \leftarrow X_1=Y_1,\ldots X_m=Y_m, p(Y_1,\ldots,Y_m); \\
& \qquad\qquad p \in \Pi,\ p \neq '='\}
\end{aligned}
$$

An equational theory is a first order theory where equations are inferred from a set of equations E with the axioms E_I, E_S, E_T, and $E_f (f \in \Sigma)$. An E-unifier for two terms s and t is a substitution θ such that $E \cup EQ \models s\theta = t\theta$, where $\models$ denotes

the logical implication. E-unification is to find E-unifiers. An ordinal unifier is a substitution θ such that $EQ \models s\theta = t\theta$.

The standard PROLOG system unifies two words in each step of refutations. The extension of unification in logic programming is to replace the unifiers by E-unifiers. Logically speaking, for a given goal clause $\leftarrow A_1,\ldots,A_k$, the refutation of PROLOG computes a substitution θ such that

$$P \cup EQ \models \forall ((A_1 \wedge \ldots \wedge A_k)\theta),$$

while the refutation with E-unification computes one such that

$$P \cup E \cup EQ \models \forall ((A_1 \wedge \ldots \wedge A_k)\theta),$$

where $\forall (F)$ is the universal closure of a formula F. The mgu of two terms is unique up to variants (i.e., renaming variables), but the most general E-unifiers of two terms are not unique. Thus when we introduce the generalized unification in logic programming, we consider a complete set of E-unifiers.

Let U be a set of E-unifiers of two terms, then U is complete if it satisfies the following two conditions:

1) For any element σ of U,

$$D(\sigma) \subset (V(s) \cup V(t)),$$
$$I(\sigma) \cap (V(s) \cup V(t)) = \phi .$$

2) For each E-unifier θ of s and t, there exist an element σ of U and a substitution γ such that

$$E \cup EQ \models X\theta = X\sigma\gamma$$

for each variable X in $V(s) \cup V(t)$.

Thus, one may imagine that we can implement the extended refutation by simply replacing the algorithm for mgu by that for complete set. Some researchers have introduced a congruence relation of ground atoms, given the fixpoint semantics of logic programs and shown the completeness of refutations (Goguen and Meseguer [3], Jaffer et al. [7]). Their theory is for the system that the algorithm for a complete set is a built-in procedure of the refutation procedure. Thus they left the control of the algorithm for the complete sets out of considerations. In implementing a system by using E-unification, we need to control the searches for the SLD-refutation and for the complete set, because the complete set may be infinite.

For example, consider the following logic program:

P={ C_1: part([],B,[],[])←

C_2: part([A|D],B,[A|X],Y)← A >= P,part(D,P,X,Y)

C_3: part([A|D],B,X,[A|Y])← A < P, part(D,P,X,Y)}.

We choose the set of equations in Example 2.1, replace the mgu algorithm by the complete set algorithm, and try to refute the goal clause:

G: ←part([3,7,5],5,app(W,[L]),Z).

We intend that W will be substituted by the list which contains the numbers more than or equal to 5 and lacks the last element. At first, the system unifies the body of G and the head of C_2. Consequently, each variables in C_2 is substituted by some term, and the new goal clause is derived. For example, the following goal is derived:

G_1: ←3>=5, part([7,5],5,[3],Y).

Then no refutation of G_1 succeeds, and the system backtracks. Since the complete sets of E-unifiers are infinite, the system may find another E-unifiers of the body of G and the head of C_2, and derive another goal clause. Whatever goal clause may be derived, no refutation for the goal succeeds because 3>=5 is always false. Thus we need to control the E-unification so that the system may unify the body of G and the head of C_3. In standard PROLOG, the following goal clause corresponds to G:

←p([3,7,5],5,V,Z),append(W,[L],V).

Thus the system should unify app(W,[L]) and [A|X] after the refutation for G_1.

Such a refutation cannot obtained simply by replacing mgu's by complete sets, and so the SLD-resolution and E-unification are to be treated on the same level. Since EQ and E are sets of definite clauses, SLD-resolution can include the inference of E-unification. Then the next problem is how to treat the clauses in EQ. In theorem proving, in order to treat equations, the term rewriting system and the paramoduration are used for replacing the clauses which may fall into infinitely looping. In this paper, we use narrowing, which combines two procedures, and has been introduced to enumulate the complete set of E-unifiers (Hullot [6]). In discussions on such inference rules, E-interpretations have been used instead of EQ. But in this paper, we use EQ IN order to discuss the fixpoint semantics.

3. Term Rewriting System and Narrowing

In this section we discuss the term rewriting systems and narrowing according to Hullot [6]. We denote the position of a subword in a word by a sequence of positive numbers. The positive numbers in a sequence are separated by periods '.' from each other. The empty sequence is denoted by Λ .

Definition 3.1. Let s be a word or a sequence of words. The occurrence of a subterm of s is the sequence of positive numbers defined by the following rules:

1) The occurrence of s is Λ .

2) If s is a word $f(t_1,\ldots,t_n)$ and its occurrence is u, then the occurrence of t_i is u.i.

3) If s is a sequence of words $t_1,\ldots,t_n$ and its occurrence is u, then the occurrence of t_i is u.i.

Suppose each of s and t is a word or a sequence of a word, s/u denotes the subword of s whose occurrence is u, and $s[u\leftarrow t]$ denotes the word or the sequence which is obtained by replacing subword s/u by t. The symbol Oc(s) denotes the set of occurrences of all subterms of s, and Og(s) denotes the set of occurrences of all non-variable subterms of s.

Definition 3.2. A term rewriting system R is a set of pairs of terms such that each element $\langle \alpha , \beta \rangle$ satisfies the following conditions:

1) $V(\beta) \subset V(\alpha)$,

2) $\alpha \notin V$.

The element of R is called a rewriting rule, and denoted by $\alpha \rightarrow \beta$.

Definition 3.3. Let R be a term rewriting system. We define a binary relation $\rightarrow_R$ on $T(\Sigma \cup V)$ by:

$t \rightarrow_R s$

iff there exist $u \in Oc(t)$, $\alpha \rightarrow \beta \in R$, and a substitution σ such that $t/u = \alpha\sigma$ and $s = t[u \leftarrow \beta\sigma]$.

We call $\rightarrow_R$ a reduction relation on R. The reflexive transitive closure of relation $\rightarrow_R$ is denoted by $\overset{*}{\rightarrow}_R$.

Definition 3.4. Let R be a term rewriting system. R is confluent if for all t, t_1 and t_2 such that $t \overset{*}{\rightarrow}_R t_1$ and $t \overset{*}{\rightarrow}_R t_2$, there exists a term such that $t_1 \overset{*}{\rightarrow}_R s$ and $t_2 \overset{*}{\rightarrow}_R s$. R is confluent with respect to ground terms if this condition is satisfied in case t is a ground term.

Definition 3.5. Let R be a term rewriting system. R is finitely terminating if there exists no infinite derivation $t_1 \rightarrow_R t_2 \rightarrow_R t_3 \rightarrow_R \cdots$. R is canonical if it is confluent and finitely terminating.

A term t is in normal form if there exists no term s such that $t \rightarrow_R s$. If R is canonical, each term admits a unique normal term R(t) such that $t \overset{*}{\rightarrow}_R R(t)$. We call R(t) a normal form of t. Moreover, let θ be a substitution $\{X_1 \leftarrow t_1, \ldots, X_n \leftarrow t_n\}$. If each $t_1, \ldots, t_n$ is in normal form, then θ is in normal form. The substitution $\{X_1 \leftarrow t_1, \ldots, X_n \leftarrow t_n\}$ is called a normal form of θ.

Let E be a set of equations such that each element $\alpha = \beta \leftarrow$ satisfies the conditions of Definition 3.2. Then we can regard E as a term rewriting system R(E). In this paper, we only treat such a set of equations, so we do not distinguish between E and R(E). Then if E is canonical, for each pair of terms t and s,

$$E \cup EQ \models s = t \Leftrightarrow R(t) = R(s).$$

Definition 3.6. Let R be a term rewriting system. We define a binary relation $\overset{\curlyvee}{\rightarrow}_R$ on A(L) by:

$M \overset{\curlyvee}{\rightarrow}_R N$

iff there exist an occurrence $u \in Og(M)$ and a rewriting rule $\alpha \rightarrow \beta \in R$ such that $N = (M[u \leftarrow \beta])\theta$, where variables in $\alpha \rightarrow \beta$ are renamed so that $V(M) \cap V(\alpha) = \phi$.

We call $\overset{\curlyvee}{\rightarrow}_R$ a narrowing relation on R.

If E is canonical, the complete set of E-unfiers can be enumulated by iteration of narrowing and resolution with the

identity axiom E_I ([6]).

Narrowing resembles the procedure of SLD-resolution, so it is easy to add narrowing to PROLOG system in order to treat functional expressions. For this purpose, we need to clarify the semantics of narrowing by using Herbrand model.

4. Equations and Logic Programming

In this section, we introduce a new refutation system which consists of narrowing and SLD-resolutions. Since narrowing is an inference of equational theories that dose not have the inference rules E_p ($p \in \Pi$), we need some device to treat predicates. We do not treat negative information here. Thus we make use of the homogeneous form of definite clauses in van Emden and Lloyd [16].

Definition 4.1.

1) A homogeneous definite clause is the definite clause of the form

$$p(X_1,\ldots,X_m) \leftarrow B_1,\ldots,B_n$$

where p is not the symbol '=', $m,n \geq 0$, and $X_1,\ldots,X_m$ are distinct variables.

2) The homogeneous form of a definite clause

$$p(t_1,\ldots,t_m) \leftarrow B_1,\ldots,B_n$$

is the clause

$$p(X_1,\ldots,X_m) \leftarrow X_1=t_1,\ldots,X_m=t_m,B_1,\ldots,B_n$$

where $X_1,\ldots,X_m$ are distinct variables not appearing in the original clause.

3) The homogeneous form of a logic program is the collection of homogeneous forms of each of its clause.

By transforming a definite clause into its homogeneous form, we add equations representing unification to the body of the original clause. The transformation is justified by the following lemma. Thus we treat only homogeneous definite clauses.

Lemma 4.1. Let P be a program and P' be the homogeneous

form of P. Then P∪ E∪ EQ and P'∪ E∪ EQ are logically equivalent.

Example 4.1. Let

P= { member(X,[X|Y])←
member(X,[Y|Z])← member(X,Z) }.

Then the homogeneous form of P is given by the following program:

{ member(X_1,X_2)← X_1=X,X_2=[X|Y]
member(X_1,X_2)← X_1=X,X_2=[Y|Z],member(X,Z) }.

According to Proposition 1 of [16], E_I is the only clause in EQ that is necessary to treat the program of homogeneous form in PROLOG, while Hullot [6] has shown that E_I is also the only clause in EQ for E-unification with narrowing. Thus when we consider a system that has both narrowing and SLD-resolution, we may choose only E_I as far as all the definite clauses except the equations are homogeneous. In the following sections, we construct an Herbrand model of such a program, and we discuss the completeness of refutation to show that the above choice is sufficient under some conditions. For the purpose, we first need a definition of a program consisting of definite clauses and equations.

Definition 4.2. A program Pr is a set of clauses consisting of three parts E, P and {X=X← }, where E is a set of equations, P is a set of homogeneous definite clauses, and {X=X← } is a singleton set of E_I. E(Pr) and P(Pr) denote E and P∪ {X=X← } respectively.

Note that only the clause X=X← is the clause in which the predicate symbol '=' occurs.

Example 4.2. Let

Pr= { app([],X)=X←
app([A|X],Y)=[A|app(X,Y)]←
member(X_1,X_2)← X_1=X,X_2=[X|Y]
member(X_1,X_2)← X_1=X,X_2=[Y|Z],member(X,Z)

$X=X\leftarrow$ }.

Then

E(Pr)= { app([],X)=X←
app([A|X],Y)=[A|app(X,Y)]← },
P(Pr)= { member(X_1,X_2)← X_1=X,X_2=[X|Y]
member(X_1,X_2)← X_1=X,X_2=[Y|Z],member(X,Z)
X=X← }.

Now we define a derivation procedure as an extension of standard PROLOG, where narrowing and SLD-resolution are treated equally.

Definition 4.3. A goal clause is a clause of the form

$$\leftarrow A_1,\ldots,A_k.$$

The predicate symbol of A_i may possibly be the symbol '='. The occurrence of the word occurring in the goal clause is that in the sequence of atoms $A_1,\ldots,A_k$.

From now on, when a variant of a clause C is an element of a set S, we write $C\in S$.

Definition 4.4. Let Pr be a program, and G be a goal. A derivation sequence for $Pr\cup\{G\}$ is a (finite or infinite) sequence of quadruplets $\langle G_i,\theta_i,C_i,u_i\rangle$ $(i=0,1,2,\ldots)$ which satisfies following conditions:

1) G_i is a goal clause, θ_i is a substitution, C_i is a variant of a clause in Pr, u_i is an occurrence in $Og(G_{i-1})$, and G_0 is G.

2) The variables in C_i is standardized apart, i.e.,

$$V(C_i)\cap(\cup_{j<i}(V(\langle G_j,\theta_j,C_j,u_j\rangle))=\phi.$$

3) In case G_i/u_{i+1} is an atom, $C_{i+1}\in P(Pr)$. If C_{i+1} is $A\leftarrow B_1,\ldots,B_q$, G_i is $\leftarrow A_1,\ldots,A_k$, and $G_i/u_{i+1}=A_j$, then θ_{i+1} is an mgu of A and A_j, and G_{i+1} is the goal

$$\leftarrow(A_1,..,A_{j-1},B_1,...,B_q,A_{j+1},..,A_k)\theta_{i+1}.$$

4) In case G_i/u_{i+1} is a term, $C_{i+1}\in E(Pr)$. If C_{i+1} is $\alpha=\beta\leftarrow$, then θ_{i+1} is an mgu of α and G_i/u_{i+1}, and G_i is the goal

$$(G_i[u_{i+1}\leftarrow\beta])\theta_{i+1}.$$

In the definition above, condition 3) represents resolution with a clause in P(Pr), and 4) represents narrowing using the equation in E(Pr). An equation in a goal clause can be resolved only with the clause X=X← . There are no restrictions at all for the selection of u_i's. The rule for this selection is called a computation rule, and is important in implementing the system for the derivation.

A goal H is derived from Pr∪ {G} if there exists a finite derivation $\{<G_i, \theta_i, C_i, U_i>\}_{i\leq n}$ for Pr∪ {G} such that $G_n = H$.

Definition 4.5. A refutation for Pr∪ {G} is a finite derivation for Pr∪ {G} which derives the empty clause □ .

Example 4.3. Let Pr be the program in Example 4.2. Let G be the goal clause ←member(1, app(X,[2])). Then a refutation for Pr∪ {G} is illustrated by the following figure, where the underlined words are selected by u_i's.

$\underline{\leftarrow member(1,app(X,[2]))}$
| $member(X_{11},X_{21}) \leftarrow X_{11}=X_1, X_{21}=[X_1|Y_1]$
$\leftarrow \underline{1=X_1}, app(X,[2])=[X_1|Y_1]))$
| $X_2=X_2 \leftarrow$
$\leftarrow \underline{app(X,[2])}=[1|Y_1]$
| $app([A_3|X_3],Y_3)=[A_3|app(X_3,Y_3)] \leftarrow$
$\leftarrow [A_3|\underline{app(X_3,[2])}]=[1|Y_1]$
| $app([],X_4)=X_4 \leftarrow$
$\leftarrow \underline{[A_3,2]=[1|Y_1]}$
| $X_5=X_5 \leftarrow$
□

We can regard each refutation as a computation, so that its result is the substitution which is obtained by composing all the substitutions used in the refutation.

Definition 4.6. Let Pr be a program, G a goal, and $\{<G_i, \theta_i, r_i, u_i>\}_{i\leq n}$ be a refutation for Pr∪ {G}. Then a substitution $\theta = (\theta_0 \ldots \theta_n)|_{V(G)}$ is called a computed answer substitution for Pr∪ {G}.

Strictly speaking, a refutation for $Pr \cup \{G\}$ is that for $Pr \cup EQ \cup \{G\}$, and an answer substitution for $Pr \cup \{G\}$ is that for $Pr \cup EQ \cup \{G\}$.

Example 4.4. The computed answer substitution for the refutation in Example 4.3 is $\theta = \{X \leftarrow [1|X_1]\}$.

The following theorem guarantees the soundness of the computed answer substitution given by a refutation.

Theorem 4.1. Let Pr be a program and G be a goal clause

$$\leftarrow A_1, \ldots, A_n.$$

Then every computed answer substitution θ for $Pr \cup \{G\}$ is a correct answer substitution, i.e.,

$$Pr \cup EQ \models \forall ((A_1 \wedge \ldots \wedge A_n)\theta).$$

Proof. Let $\{\langle G_i, \theta_i, C_i, u_i \rangle\}_{i=1}^{n}$ be a derivation for $Pr \cup \{G\}$. The theorem is proved by induction on n, the length of refutation.

Suppose $n = 1$. Then G is a goal of the form $\leftarrow A_1$, and $A_1\theta_1 = A\theta_1$ for a unit clause $A \leftarrow$. Thus, $Pr \cup EQ \models \forall (A_1\theta_1)$.

Now suppose the theorem holds for n-1, and consider the case that the length is n.

In case G/u_1 is an atom, we can prove that

$$Pr \cup EQ \models \forall ((A_1 \wedge \ldots \wedge A_k)\theta_1 \ldots \theta_n)$$

because the first step of the derivation is SLD-resolution. Then we consider the case that G/u_1 is a term. Let C_1 be the equation $\alpha = \beta \leftarrow$, and $G_1/u_1 = t$. Then G_1 is the goal

$$\leftarrow ((A_1, \ldots, A_n)[u_1 \leftarrow \beta])\theta_1,$$

and t is a subterm of A_m. We can put $u_i = m.v$ where $v \neq \Lambda$. Since $Pr \models \forall (\alpha\theta_1 = \beta\theta_1)$, and $\alpha\theta_1 = t\theta_1$,

$$EQ \models \forall (t\theta_1 = \beta\theta_1 \wedge A_m[v \leftarrow \beta]\theta_1 \rightarrow A_m\theta_1).$$

Thus

$$Pr \cup EQ \models \forall ((A_1 \wedge \ldots \wedge A_k)\theta_1 \ldots \theta_n).$$

5. Fixpoint Semantics

In this section we discuss the fixpoint semantics of

programs defined in the previous section. In general, the fixpoint semantics of a program is given by the least fixpoint of a function on a complete lattice. In logic programming, the fixpoint coincides with an Herbrand model by selecting the power set of Herbrand base as the complete lattice. Precisely, suppose P be a logic program (in the ordinal sense). We define a mapping $T_P: 2^{B(P)} \to 2^{B(P)}$ by

$$T_P(I) = \{A \in B(L): A \leftarrow B_1, \ldots, B_n \text{ is a ground instance of } P \text{ and } \{B_1, \ldots, B_n\} \subset I\}.$$

Then the fixpoint semantics of P is given by the least fixpoint of T_P. Since P is a set of definite clauses, P has a least Herbrand model M(P), which is characterized by logical consequences. That is,

$$M(P) = \{A \in B(L): P \models A\}.$$

The characteristic aspect of logic programming is that the least fixpoint of T_P coincides with M(P).

Now we turn our discussion to equational theories. As we have shown in Section 2, the equational axioms are expressed in the form of definite clauses EQ, and a set of equations E is also a definite clause. Then $E \cup EQ$ has its least model $M(E \cup EQ)$ and

$$M(E \cup EQ) = \{s=t;\ s,t \in T(\Sigma) \text{ and } E \cup EQ \models s=t\}.$$

Now we define the same mapping for narrowing as for SLD-resolution, and investigate the relation between its fixpoint and $M(E \cup EQ)$.

Let E be a set of equations. Then we define a mapping $N_E: 2^{B(P)} \to 2^{B(P)}$ by

$$N_E(I) = \{p(t_1, \ldots, t_m) \in B(L);\ p(t_1, \ldots, t_m) \in I \text{ or } t_i \to_E u_i \text{ and } p(t_1, \ldots, s_i, \ldots, t_m) \in I \text{ for some argument } t_i \text{ and term } u_i\}.$$

Lemma 5.1. N_E is continuous.

The proof of the lemma is analogue to that of T_P.

We define inductively the set $N_E \uparrow k$ for every positive integers k as follows:

$$N_E \uparrow 1 = Id, \text{ where } Id = \{s=s;\ s \in T(\Sigma)\},$$
$$N_E \uparrow (k+1) = N_E(N_E \uparrow k)).$$

Id is the least model for $\{X=X\leftarrow\}$ and also is that of EQ. Moreover we define

$$N_E\uparrow \omega = \cup_{k\in\omega} (N_E\uparrow k).$$

$N_E\uparrow \omega$ is the least of the fixpoints containing Id, and holds the following property :

$$N_E\uparrow \omega = \{s=t;\ s,t\in T(\Sigma) \text{ and there exists } u\in T(\Sigma) \text{ such that } s \overset{*}{\rightarrow}_E u \text{ and } t \overset{*}{\rightarrow}_E u\} \quad (5.1)$$

From the property, it holds that $s=t\in N_E\uparrow \omega$ implies $E\cup EQ\models s=t$, that is, $N_E\uparrow \omega \subset M(E\cup EQ)$.

Lemma 5.2. $N_E\uparrow \omega = M(E\cup EQ)$ iff E is confluent with respect to ground terms.

Now we are in the position to discuss the fixpoint semantics of a program Pr defined by Definition 4.2. Since Pr is a set of definite clauses, there exists the least Herbrand model $M(Pr\cup EQ)$. In the refutation procedure by Definition 4.4, narrowing is the procedure to treat the equations in E(Pr), and SLD-resolution is that for the definite clauses in P(Pr). Thus, to denote the refutation procedure, we can define a mapping $S_{Pr}: 2^{B(P)}\rightarrow 2^{B(P)}$ by:

$$S_{Pr}(I) = N_{E(Pr)}(I)\cup T_{P(Pr)}(I).$$

Lemma 5.3. S_{Pr} is continuous.

Now we define the set $S_{Pr}\uparrow k$ for non-negative integers inductively.

$$S_{Pr}\uparrow 0 = \phi\ ,$$
$$S_{Pr}\uparrow (k+1) = S_{Pr}(S_{Pr}\uparrow k).$$

Let

$$S_{Pr}\uparrow \omega = \cup_{k\in\omega} (S_{Pr}\uparrow k).$$

S_{Pr} is the least fixpoint of S_{Pr}. For the sake of convenience, we define the following sets:

$$E(S_{Pr}\uparrow \omega) = \{s=t;\ s=t\in S_{Pr}\uparrow \omega\},$$
$$P(S_{Pr}\uparrow \omega) = S_{Pr}\uparrow \omega - E(S_{Pr}\uparrow \omega).$$

The following lemma justifies the restriction on the definite clause.

Lemma 5.4. Let Pr be a program. Suppose $E(S_{Pr}\uparrow \omega) = M(E(Pr) \cup EQ)$. Then $p(t_1,\ldots,t_m) \in P(S_{Pr}\uparrow \omega)$ and $\{t_1=s_1,\ldots, t_m=s_m\} \subset E(S_{Pr}\uparrow \omega)$ implies $p(s_1,\ldots,s_m) \in P(S_{Pr}\uparrow \omega)$.

Proof. We prove the lemma by induction on n such that $p(t_1,\ldots,t_m) \in S_{Pr}\uparrow n$.

Suppose first that $n=1$. Then $p(t_1,\ldots,t_m) \in S_{Pr}\uparrow 1$ means that the clause $p(X_1,\ldots,X_m)\leftarrow$ is contained by Pr. Because the predicate symbol is not '=' and $X_1,\ldots,X_m$ are distinct variables, it holds that $p(s_1,\ldots,s_m) \in S_{Pr}\uparrow 1$.

Now suppose the lemma holds for n-1, and let $p(t_1,\ldots,t_m) \in S_{Pr}\uparrow n$. By the definition of S_{Pr}, there are two cases to be considered.

Case 1. $p(t_1,..,t_m) \in T_{P(Pr)}(S_{Pr}\uparrow (n-1))$.

There exists a clause

$$p(X_1,\ldots,X_m)\leftarrow B_1,\ldots,B_k$$

such that

$$p(X_1,\ldots,X_m)\theta = p(t_1,\ldots,t_m),$$
$$\{B_1\theta,\ldots,B_k\theta\} \subset S_{Pr}\uparrow (n-1)$$

for some θ. Let

$$\tau = \{X_1\leftarrow t_1,\ldots,X_m\leftarrow t_m\},$$
$$\sigma = \{X_1\leftarrow s_1,\ldots,X_m\leftarrow s_m\}.$$

Note that $\theta = \tau \cup \zeta$ for some ground substitution ζ. Let

$$B_i\theta = q_i(u_1,\ldots,u_{mi}),$$
$$B_i(\sigma \cup \zeta) = q_i(v_1,\ldots,v_{mi}).$$

Then

$$\{u_1=v_1,\ldots,u_{mi}=v_{mi}\} \subset E(S_{Pr}\uparrow \omega) \qquad (5.2)$$

by the assumption of the lemma. In case q_i is '=', $B_i(\sigma \cup \zeta) \in S_{Pr}\uparrow \omega$ by (5.2). In case q_i is not '=', $B_i(\sigma \cup \zeta) \in S_{Pr}\uparrow \omega$ by the induction hypothesis and (5.2). Thus $p(s_1,\ldots,s_m) \in S_{Pr}\uparrow \omega$.

Case 2. $p(t_1,\ldots,t_m) \in N_{E(Pr)}(S_{Pr}\uparrow (n-1))$.

There exists an ground term u_i such that

$$t_i \rightarrow_E u_i,$$
$$p(t_1,..,u_i,..,t_m) \in S_{Pr}(n-1).$$

From the assumption of the lemma,

$$u_i=s_i \in E(Pr),$$

which implies

$$p(s_1,\ldots,s_m) \in S_{Pr}\uparrow \omega$$

by the induction hypothesis.

We combine Lemma 5.3 and 5.4 into a corollary:

Corollary 5.1. Let Pr be a program, and let E(Pr) be confluent with respect to ground terms. Then $p(t_1,\ldots,t_m) \in P(S_{Pr}\uparrow \omega)$ and $\{t_1=s_1,\ldots,t_m=s_m\} \in E(S_{Pr}\uparrow \omega)$ implies $p(s_1,\ldots,s_m) \in P(S_{Pr}\uparrow \omega)$.

We can prove the following theorem about the relation between the fixpoint and least Herbrand model.

Theorem 5.1. Let Pr be a program. If E(Pr) is confluent with respect to ground terms, then $S_{Pr}\uparrow \omega = M(Pr \cup EQ)$.

Proof. ($\supset$) Since E(Pr) is confluent with respect to ground terms, $S_{Pr}\uparrow \omega$ is a model for EQ by the Lemma 5.2 and Corollary 5.5. Thus $T_{P(Pr)}(S_{Pr}\uparrow \omega) \subset S_{Pr}\uparrow \omega$ implies that $S_{Pr}\uparrow \omega$ is a model for Pr.

($\subset$) By Lemma 5.2, $t=s \in S_{Pr}\uparrow \omega$ implies $Pr \cup EQ \models s=t$.

Next we prove on n that $p(t_1,\ldots,t_m) \in S_{Pr}\uparrow n$ implies $Pr \cup EQ \models p(t_1,\ldots,t_m)$, where p is not '='.

First suppose n= 1. Then $p(t_1,\ldots,t_m) \in S_{Pr}\uparrow 1$ implies that there exists a clause $p(X_1,\ldots,X_m) \leftarrow$ in Pr. Thus $Pr \cup EQ \models p(t_1,\ldots,t_m)$.

Now suppose the theorem holds for n-1 and $p(t_1,\ldots,t_m) \in S_{Pr}\uparrow n$. In case $p(t_1,\ldots,t_m) \in T_{P(Pr)}(S_{Pr}\uparrow (n-1))$, there exist a clause $p(X_1,\ldots,X_m) \leftarrow B_1,\ldots,B_k$ and a substitution θ such that

$$p(X_1,\ldots,X_m)\theta = p(t_1,\ldots,t_k),$$
$$\{B_1\theta,\ldots,B_m\theta\} \subset S_{Pr}\uparrow (n-1).$$

Then $Pr \cup EQ \models p(t_1,\ldots,t_m)$ by the induction hypothesis. In case $p(t_1,\ldots,t_m) \in N_{E(Pr)}(S_{Pr}\uparrow (n-1))$, there exists a ground term u_i such that

$$t_i \rightarrow_{E(Pr)} u_i,$$
$$p(t_1,..,u_i,..,t_m) \in S_{Pr}\uparrow (n-1).$$

Since $Pr \cup EQ \models t_i=u_i$, $Pr \cup EQ \models p(t_1,\ldots,t_m)$ by the induction hypothesis and the axiom E_I and E_p in EQ.

6. Completeness of Refutation

In this section, we show the completeness of the refutation along the discussions in Lloyd [10].

The success set of a program Pr is the set

$\{A \in B(L);$ there exists a refutation for $Pr \cup \{\leftarrow A\}\}$.

We must discuss the soundness and completeness for a refutation from the three points, 1) answer substitution, 2) unsatisfiablity and 3) the success set and the least Herbrand model.

The soundness for computed answer substitution is shown by Theorem 4.1, and reduces the soundness for unsatisfiablity and success set by Theorem 5.1.

Lemma 6.1. Let Pr be a program and G be a goal clause. If there exists a refutation for $Pr \cup \{G\}$, then $Pr \cup EQ \cup \{G\}$ is unsatisfiable.

Corollary 6.1. The success set for a program Pr is contained by $M(Pr \cup EQ)$.

We discuss the completeness for refutations. In the discussion, we need to treat mgu's carefully. In practice, we assume that mgu's are computed by the Algorithm 1 in Martelli and Montaneri [12]. Such an mgu θ has the property that

$$D(\theta) \cap I(\theta) = \phi ,$$

$$D(\theta) \cup I(\theta) \subset V(s) \cup V(t).$$

In order to show the completeness of refutations for programs Pr and goals, we require that E(Pr) is canonical. When E(Pr) is canonical, each term and substitution admits its normal form. At first, we prepare a lemma about the relation between a normal substitution and an mgu.

Lemma 6.2. Let s and t be two words, and η be a ground-and-normalized substitution such that $D(\eta) \cap V(t) = \phi$. If there exists an mgu θ of $s\eta$ and t, then there exist an mgu μ of s and t and a ground-and-normalized substitution ζ such that

$$\eta \theta = \mu \zeta .$$

Using the above lemma, we can prove the lifting lemma, where lifting for narrowing is the same as the "projection of narrowing" in Kanamori [8]. We put the restriction that each substitution η for lifting is ground since we prove the completeness of refutation using Herbrand model. Then we can combine lifting for narrowing and SLD-resolution without taking care of the set $I(\eta)$.

Lemma 6.3.(Lifting lemma, Robinson and Kanamori) Let Pr be a program such that E(Pr) is canonical, G be a goal clause, and η be a ground-and-normalized substitution. If there is a refutation for Pr$\cup$ {Gη }, then there exists a refutation for Pr $\cup$ {G}. Furthermore, if θ and μ are the computed answer substitutions for Pr$\cup$ {Gη } and for Pr$\cup$ {G}, respectively, then there exists a substitution γ such that

$$\eta\ \theta\ |_{V(G)} = \mu\ \gamma\ |_{V(G)}.$$

From the lifting lemma , we can prove the following theorem about the relation between the success set and the fixpoint semantics.

Theorem 6.1. Let Pr be a program such that E(Pr) is canonical. Then $S_{Pr}\uparrow\omega$ is contained by the success set of Pr.

Proof. Suppose $s=t \in E(S_{Pr}\uparrow\omega)$. By the note (5.1) and the fact that X=X$\leftarrow$ is the only clause that has the symbol '=' in its head, there exists a ground term u such that $s \overset{*}{\rightarrow}_{E(Pr)} u$, $t \overset{*}{\rightarrow}_{E(Pr)} u$. Thus there exists a refutation for Pr$\cup$ {$\leftarrow$ s=t}, by the condition 1) of Definition 3.2.

To prove that the success set implies $P(S_{Pr}\uparrow\omega)$, we prove the following assertion by the induction on n such that $p(t_1,\ldots,t_m) \in S_{Pr}\uparrow n$.

Assertion. If $p(t_1,\ldots,t_m) S_{Pr}\uparrow n$, then for all ground terms $s_1,\ldots,s_m$ such that $t_1 \overset{*}{\rightarrow}_{E(Pr)} s_1,\ldots,t_m \overset{*}{\rightarrow}_{E(Pr)} s_m$, $p(s_1,\ldots,s_m)$ is an element of success set.

Suppose first that n= 1. Then $p(t_1,\ldots,t_m)$ implies that Pr

has a clause $p(X_1,\ldots,X_m)\leftarrow$. Thus the assertion holds for n= 1. Now suppose the assertion holds for n-1, and $p(t_1,\ldots,t_m)\in S_{Pr}\uparrow n$.

Case 1. $p(t_1,\ldots,t_m)\in T_{P(Pr)}(S_{Pr}\uparrow (n-1))$.

There exists a clause of the form $p(X_1,..,X_m)\leftarrow B_1,..,B_k$ such that

$$p(t_1,..,t_m)= p(X_1,..,X_m)\theta\ ,$$
$$\{B_1\theta\ ,\ldots B_k\theta\ \}\subset S_{Pr}\uparrow (n-1)$$

for some θ . We can put $\theta = \tau \cup \sigma$ where $\tau = \{X_1\leftarrow t_1,\ldots,\ X_m\leftarrow t_m\}$. Let the normal form of each t_i be u_i, then the normal form of τ is the substitution $\mu = \{X_1\leftarrow u_1,\ldots,X_m\leftarrow u_m\}$. Let η be the normal form of σ . Then since $B_i\theta \overset{*}{\rightarrow}_{E(Pr)} B_i(\mu \cup \eta\ )$ and $B_i\theta \in S_{Pr}\uparrow (n-1)$, there exists a refutation for $B_i\theta$. Moreover, $\mu \cup \eta = \mu\ \eta$, and there exists a refutation for $B_i\mu$ by lifting lemma. Thus there exists a refutation for $\leftarrow p(u_1,\ldots,u_m)$. Since u_i is also the normal form of s_i, there exists a refutation for $\leftarrow p(s_1,\ldots,s_m)$.

Case 2. $p(t_1,\ldots,t_m)\in N_{E(Pr)}(S_{Pr}\uparrow (n-1))$.

There exist t_k and a ground term v such that $t_k\rightarrow v$ and $p(t_1,..,v,..,t_m)\in S_{Pr}\uparrow (n-1)$. Let u_i be the normal form of t_i for $i= 1,\ldots,n$. Then the normal form of v is u_k, and from the induction hypothesis there exists a refutation for $\leftarrow p(u_1,..,u_m)$. Since u_i is the normal form of s_i, there exists a refutation for $\leftarrow p(s_1,\ldots,s_m)$.

If E(Pr) is canonical, it is confluent with respect to ground terms from the condition 1) of Definition 3.2. Thus, we combine theorem 5.1 and 6.1 into a theorem to show the completeness on the success set:

Theorem 6.2. Let Pr be a program such that E(Pr) is canonical. Then the following three sets are identical.

1) M(Pr$\cup$ EQ),

2) $S_{Pr}\uparrow \omega$,

3) the success set of Pr.

Corollary 6.2. Let Pr be a program such that E(Pr) is canonical, and G be a goal clause. If Pr$\cup$ EQ$\cup$ {G} is

unsatisfiable, then there exists a refutation for Pr$\cup$ {G}.

Proof. Let G be the goal $\leftarrow A_1,\ldots,A_k$. Since P$\cup$ EQ$\cup$ {G} is unsatisfiable, G is false in M(Pr$\cup$ EQ). Hence some ground instance $G\theta$ of G is true in M(Pr$\cup$ EQ). Thus $\{A_1\theta,\ldots,A_k\theta\} \subset$ M(Pr$\cup$ EQ). Let η be the normal form of θ. Then, by Lemma 5.5, $\{A_1\eta,\ldots,A_k\eta\} \subset$ M(Pr$\cup$ EQ). By Theorem 6.2, there exists a refutation from $\leftarrow A_i\eta$ for $i=1,\ldots,k$. Since each $A_i\eta$ is ground, we can combine these refutations into a refutation of $G\eta$. Finally we can apply the lifting lemma and we can get a refutation of G.

Let R be a confluent (finitely terminating) term rewriting system over $T(\Sigma \cup V)$, and we make the function symbol set Σ' by adding some new constant symbols to Σ. Then R is confluent (finitely terminating) as a term rewriting system over $T(\Sigma' \cup V)$. The terms in normal form are in normal form after the extension of Σ. Noting this point, the following theorem justifies the completeness for computed answer substitution. For the completeness, we must consider the congruence relation on E $\cup$ EQ. Thus strictly speaking, it is E-completenss.

Theorem 6.3. Let Pr be a program such that E(Pr) is canonical and G be a goal.Then there exist a computed answer substitution μ and a substitution γ such that

$$E(Pr)\cup EQ \models X\theta = X\mu\gamma$$

for each variable X in G.

Proof. Let G be the goal of the form $\leftarrow A_1,\ldots,A_k$, and η be the normal form of θ. Since θ is the correct answer substitution, η is correct, i.e.,

$$Pr\cup EQ \models \forall((A_1 \wedge \ldots \wedge A_k)\eta).$$

Suppose $V((A_1 \wedge \ldots \wedge A_k)\eta) \cup I(\eta) = \{X_1,\ldots,X_n\}$. Then we make Σ' by adding new constant symbols $c_1,\ldots,c_n$ to Σ. E(Pr) is canonical as a term rewriting system over $T(\Sigma' \cup V)$. Let $\tau = \{X_1 \leftarrow c_1,\ldots,X_n \leftarrow c_n\}$. Then

$$Pr\cup EQ \models A_j\eta\tau,$$

$$Pr\cup EQ\cup(\cup_{i=1} \{c_i=c_i\}) \models A_j\eta\tau$$

for each $j=1,\ldots,n$. Since $\eta\ \tau$ is ground-and-normalized, we can construct a refutation of $\leftarrow(A_1,\ldots,A_k)\eta\ \tau$ and a refutation $\leftarrow A_1,\ldots,A_k$ in the same way as Corollary 6.2. Let σ be the answer substitution of this refutation. Then

$$\eta\ \tau\ |_{V(G)} = \sigma\ \gamma\ '|_{V(G)}.$$

We can assume that the variables $X_1,\ldots,X_n$ do not appear in the refutation and

$$D(\sigma\ \gamma\ ')\cap\{X_1,\ldots,X_n\}=\phi\ .$$

Thus by textually replacing c_i by X_i $(i=1,\ldots,n)$ equation, we can obtain that

$$\eta\ |_{V(G)} = \sigma\ \gamma\ |_{V(G)}.$$

7. An Example of Implementation

We can implement the above system by improving standard PROLOG in the following way.

1) Each definite clause is transformed into its homogeneous form.

2) The equations in a program are treated as rewriting rules.

3) The axiom E_I always exists in a program.

4) The system can select not only an atom but also a term from each goal clause.

Since we adopt a computation rule such that the left most atom is always selected, we introduce the 'del' operator in order to change the order of the equations in a goal. We illustrate the execution of the 'del' operator by an example. In the examples, we use the notation of DEC-10 PROLOG. After finding a refutation from a clause, the system returns the instance of the goal clause by the computed answer substitution.

Example 7.1. Let us give the program in Section 2 to the system.

```
?-[user].
  part([],P,[],[]).
  part([A|D],P,[A|X],Y):-A>=P,part(D,P,X,Y).
```

```
part([A|D],P,X,[A|Y]):-A<P,part(D,P,X,Y).
```

yes

Then the system cannot find the refutation for the following goal under the above computation rule.

```
?-part([3,7,5],5,app(X,[5]),Y).
```

Thus we add the operator 'del' to the goal so that the system can find the refutation under the above computation rule.

```
?-part([3,7,5],5,del(app(X,[5])),Y).                    (7.1)

part([3,7,5],5,del(app([7],[5])),[3])

yes
```

The system derives the following goal after the the third step of derivation for (7.1).

```
?-eq(del(app(X_14,[5])),[3|X_31]),eq(Y_14,Y_31),3>5,
part([7,5],5,X_31,Y_31).                                (7.2)
```

Then the system transform (7.2) into (7.3).

```
?-eq(Y_14,Y_31),3>5,part([7,5],5,X_31,Y_31),
eq(app(X_14,[5]),[3|X_31]).                             (7.3)
```

Thus the system dose not fall into infinite loops in refuting the goal (7.1).

8. Conclusion

We have introduced a refutation consisting of narrowing and SLD-resolution, have given semantics for programs, and have discussed the completeness of the refutation for programs and goals. Expressing programs and unification in a flat form

enables us to introduce the refutation which is the bases of logic programming.

In the present paper, we have not discussed computation rules, which is the important point for an implementation. In fact, to replace the mgu in PROLOG by E-unifiers is to give a particular restriction for computation rule to the above refutation. Thus we should have clarified the use of computation rules.

Comparing our system with the system for conditional equational theories [8], our system does not require the confluency for predicates which enables us to discuss the model theoretical completeness.

It does not seem so difficult to implement the extension of the system above by introducing conditional equations as proposed in [3]. However there will be some other problems of such an implementation. Especially, the finitely terminating property depends on the restriction of variables in rewriting rules, and is the essential to the completeness of narrowing. Thus we need to remove the restriction of variables in the rewriting rules, and give a new definition of reduction relation and terminating property associated with the set of conditional equations.

We also need to clarify the theoretical foundation of the negation as failure rule along the discussions in [7]. We conjecture that we can obtain the completions of equality theories by putting some restrictions on the sets of equations.

Acknowledgments

The author would like to thank Prof. Setsuo Arikawa for his constructive comments and encouragement. The author also wishes to express his sincere thanks to Dr. Makoto Haraguchi for many discussions on the problem of logic programming and equality.

References

[1] Apt, K.R. and van Emden, M.H.(1982): Contributions to the Theory of Logic Programming, J. ACM, 29, 841-862.

[2] Clark, K.L. (1978): Negation as Failure, in Logic and Databases, Gallaire, H. and Minker, J.(eds.), Plenum Press, 293-322.
[3] Goguen, J. and Meseguer, J. (1984): Equality, Types, Modules, and (Why not?) Generics for Logic Programming, J. Logic Programming, 1, 179-210.
[4] Huet, G. and Oppen, D.C. (1980): Equations and Rewrite Rules, A Survey, in Formal Language Theory, Book, R.(eds.), Academic Press, 349-405.
[5] Huet, G. (1980): Confluent reduction: Abstract Properties and Applications to Term Rewriting System, J. ACM, 27, 797-821.
[6] Hullot, J.M. (1980): Canonical Forms and Unification, Proc. 5th Conference on Automated Deduction, 318-334.
[7] Jaffer, J., Lassetz, J., and Maher, M.J. (1984): A Theory of Complete Logic Programs with Equality, J. Logic Programming, 1, 211-223.
[8] Kanamori, T. (1985): Computation by Meta-Unification with Constructors.
[9] Knuth, D.E. and Bendix, P.G. (1970): Simple Word Problems in Universal Algebras, in Computational Problems in Abstract Algebra, Leech, J.(eds.), Pergamon Press, 263-297.
[10] Kornfeld, W.A. (1983): Equality for Prolog, Proc. 8th IJCAI, 514-519.
[11] Lloyd, J.W. (1984): Foundation of Logic Programming, Springer-Verlag.
[12] Martelli, A. and Montaneri, U. (1982): An Efficient Unification Algorithm, ACM Trans. Prog. Lang. Syst, 4, 258-282
[13] Robinson, G.A. and Wos, L. (1969): Paramodulation and Theorem Proving in First Order Theories with Equality, Machine Intelligence 4, 135-150.
[14] Sato, M. and Sakurai, T.(1983): Qute: A Prolog/Lisp Type Language for Logic Programming, Proc. 8th IJCAI, 507-513.
[15] Siekmann, J. and Szabo, P. (1982): Universal Unification and a Classification of Equational Theories, Proc. 6th Conf. on Automated Deduction, Lecture Notes in Computer Science 138, Springer-Verlag, 369-389.
[16] van Emden, M.H. and Lloyd, J.W.(1984): A Logical Reconstruction of PrologII , J. Logic Programming, 1, 143-149.